Jun 2nd, 2025

Revision 1.0

Teller ERC4626 Vault

Prepared by Orion

contact@orionsecurity.xyz

Table of Contents

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

1. Introduction

Orion performed a targeted security assessment of Teller's smart contracts. The review aimed to uncover potential risks in the contract logic, assess the resilience of the system under common attack vectors, and provide actionable recommendations to improve the protocol's overall security.

1.1. Executive summary

A total of 11 findings were identified, 1 of them being of low severity, and 10 of them being of informational severity. The main low issue revolves around ERC-4626 compatibility, while the informational issues aim to provide actionable steps to improve the overall quality of the code.

1.2 Scope

Repository	`hhttps://github.com/teller-protocol/teller-protocol-v2`
Commit	`a874a945e4800b1b6649dd84a7698a7962854636`
Ecosystem	EVM

The audit covered the following files:

- `LenderCommitmentGroup_Pool_V2.sol`
- `LenderCommitmentGroupSharesIntegrated.sol`
- `UniswapPricingLibraryV2.sol`

1.3 Overview of findings

Our comprehensive review revealed no major functional or security concerns, with only a single low severity issue identified, and 10 informational issues to improve the overall code quality of the protocol.

Severity	Count
CRITICAL	0
HIGH	0
MEDIUM	0
LOW	1
INFO	10
TOTAL	11

2. Findings

L-01: Vault is not fully ERC-4626 compatible LOW

Vulnerability details

EIP-4626 explicitly mentions `maxDeposit()` / `maxMint()` MUST factor in both global and user-specific limits. In `LenderCommitmentGroup_Pool_V2`, `deposit()` / `mint()` will revert if the first depositor is not the `owner()`:

```
function deposit(
        uint256 assets,
        address receiver
    )
        public
        virtual
        whenForwarderNotPaused
        whenNotPaused
        nonReentrant
        onlyOracleApprovedAllowEOA
        returns (uint256 shares)
    {
       ...SNIP

        // Check first deposit conditions
        if (!firstDepositMade) {
            require(msg.sender == owner(), "FDM");
            require(shares >= 1e6, "IS");
            firstDepositMade = true;
        }

      ...SNIP
```

However, `maxDeposit()` / `maxMint()` fail to account for this, effectively breaking the expected behavior following the 4626 standard.

Impact

Informational, vault does not completely follow ERC-4626 standard.

Recommendation

Consider returning 0 if the caller for `maxDeposit()` / `maxMint()` is not the owner and `firstDepositMade` is still `false`.

I-01: `_tokenAmountDifference` parameter is never used INFO

Vulnerability details

The `_tokenAmountDifference` parameter in `liquidateDefaultedLoanWithIncentive()` is never used and can be safely removed.

Recommendation

Remove the `_tokenAmountDifference` parameter.

I-02: `constructor` can be removed in `LenderCommitmentGroupSharesIntegrated` INFO

Vulnerability details

The `constructor` is added in `LenderCommitmentGroupSharesIntegrated`, however it does not have any logic inside it.

Recommendation

Remove the constructor from `LenderCommitmentGroupSharesIntegrated`.

I-03: Internal functions should begin with underscore INFO

Vulnerability details

Some internal functions don't begin with underscore. It is recommended to make all internal functions have an underscore to follow Solidity style and improve readability. Concretely, these functions should be changed:

`LenderCommitmentGroupSharesIntegrated.sol`: - `mintShares()` - `burnShares()`

`LenderCommitmentGroup_Pool_V2.sol`: - `setLastUnpausedAt()` - `getUniswapPriceRatioForPoolRoutes()` - `getRequiredCollateral()` - `getPoolTotalEstimatedValue()` - `getTotalPrincipalTokensOutstandingInActiveLoans()`

Recommendation

Consider updating such functions to start with underscore.

I-04: Some functions are missing natspec INFO

Vulnerability details

Some functions are not commented following natspec format.

Recommendation

It is recommended to update all necessary functions to follow natspec format in order to improve readability and ease third-party integrations.

I-05: Require statements missing reason strings INFO

Vulnerability details

Some require statements are missing error reason strings. Concretely, the `require(_seconds < MAX_WITHDRAW_DELAY_TIME);` check performed in `setWithdrawDelayTime()`, and the `require(assets > 0);` check performed in `deposit()` and `mint()` are missing a revert reason.

Recommendation

Include a revert reason string in such checks.

I-06: Revert reason strings should be more specific INFO

Vulnerability description

Revert reason strings in require statements in the codebase have been updated to include extremely succint reason strings. For example:

```
require(
            principalTokenBalanceAfter == principalTokenBalanceBefore + assets,
            "TB"
        );
```

Although this allows to save some gas, it might hurt long-term if debugging is required.

Recommendation

There's three different paths which can be followed: - Upgrade solidity version to add custom errors in require statements. This allows to save gas while being explicit about the error triggered. - Add more specific error strings. While it adds some gas, it will be better to understand possible causes of future bugs. - Switch to an if-revert model with custom errors

I-07: Oracle protection for non-EOA's can be easily bypassed INFO

Vulnerability details

The main ERC4626 operations include a check to ensure that only EOA's or whitelisted contract addresses can obtain user shares. However, the `_afterTokenTransfer` does not include any check, which allows users to mint shares with an EOA and then transfer them to a contract address.

Recommendation

Although it is not a huge issue given that it is still enforced in withdrawals/redemptions that the owner is actually an EOA or a whitelisted contract, it is still recommended to consider capping token transfers to allowed contracts only.

I-08: Oracle protection won't work due to EIP-7702 INFO

Vulnerability detail

The `OracleProtectionManager` manager includes a check to ensure only allowed contracts can interact with shares:

```
function isOracleApprovedAllowEOA(address _sender) public returns (bool) {
        ...SNIP

        //smart contracts (delegate calls) are blocked if they havent registered and
        waited
        if (_sender != tx.origin && !oracle.isTimeExceeded(_sender)) {
            return false;
        }
```

As shown in the snippet, `_sender != tx.origin` is the check to determine if the caller is a contract. However, due to recent EIP-7702 incorporation in the Pectra upgrade, this check will no longer work.

Recommendation

Update the check to include a code length verification:

```
function isOracleApprovedAllowEOA(address _sender) public returns (bool) {
        ...SNIP

        //smart contracts (delegate calls) are blocked if they havent registered and
        waited
-        if (_sender != tx.origin && !oracle.isTimeExceeded(_sender)) {
+        if (tx.origin != _sender && msg.sender.code.length != 0 &&
        !oracle.isTimeExceeded(_sender)) {
            return false;
        }
```

I-09: Consider using constants for hardcoded values INFO

Vulnerability details

Some literals are used several times across the codebase. As an example, 10000, which is used for percentage computations, is used in multiple functions.

Recommendation

It is recommended to create a constant for values which are used multiple times. This will improve readability and minimize potential issues due to misplacing values.

As an example, a `BPS` constant could be created for the 10000 value used for percentages, just like it's done with `STANDARD_EXPANSION_FACTOR` or other expansion factors.

I-10: Remove unused imports INFO

Vulnerability details

The code currently includes several imports which are not used. As an example, the `UniswapPricingLibraryV2` includes several imports which are not utilized, such as `Math.sol`, `EnumerableSetUpgradeable.sol`, `MathUpgradeable.sol`, among others.

Recommendation

Consider removing all unutilized imports from the codebase.